```
//like the preorder tranversal, process 2 characters at a time
public TreeNode convert(char[] expr){
    if(expr.length == 0) return null;

    Stack<TreeNode> stack = new Stack<>();
    TreeNode root = new TreeNode(expr[0]);
    stack.push(root);

    for(int i=1;i<expr.length;i=i+2){
      TreeNode node = new TreeNode(expr[i+1]);
      if(expr[i]=='?'){
          stack.peek().left=node;
      }else{
          if(expr[i]==':'){
              stack.pop();
              while(stack.peek().right!= null){
                  stack.pop();
              }
              stack.peek.right().right=node;
          }
      }
      stack.push(node);
    }
    return root;
}



public class Solution {
   public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p,
TreeNode q) {
       if (root == null || root == p || root == q) return root;
       TreeNode left = lowestCommonAncestor(root.left, p, q);
       TreeNode right = lowestCommonAncestor(root.right, p, q);
       if(left != null && right != null)   return root;
       return left != null ? left : right;
   }
}
```

```
public class Solution{
    public List<Integer> TopView(TreeNode root){
      Queue<TreeNode> queue = new LinkedList<>();
      Queue<Integer> cols = new LinkedList<>();
      Set<Integer> set =  new HashSet<>();
      List<Integer> list = new ArrayList<>();

      queue.offer(root);
      cols.offer(0);

      while(!queue.isEmpty()){
          TreeNode current = queue.poll();
          int col = cols.poll();

          if(!set.contains(col)){
              if(col<0)
                  result.add(0,current.val);
              else result.add(current.val);
              set.add(col);
          }
          if(current.left!=null){
              queue.offer(current.left);
              cols.offer(col-1);
          }
          if(current.right!= null){
              queue.offer(current.right);
              cols.offer(col+1);
          }
      }

    }
}

// bottom view
//use   the   treemap,<Integer,   Integer>   instead   of   <Integer,
List<Integer>>
```

Max Array Input : An array of n numbers, and a number k Output : An array of n numbers where output = MAX(input, input[i + 1]..... input[i + k - 1])
example: [1,3,5,7,3,4,2,9], k=3 [5, 7, 7, 7, 4, 9, 9, 9]

用deque

```
public int[] maxSlidingWindow(int[] a, int k) {
        if (a == null || k <= 0) {
            return new int[0];
        }
        int n = a.length;
        int[] r = new int[n-k+1];
        int ri = 0;
        // store index
        Deque<Integer> q = new ArrayDeque<>();
        for (int i = 0; i < a.length; i++) {
            // remove numbers out of range k
            while (!q.isEmpty() && q.peek() < i - k + 1) {
                q.poll();
            }
            // remove smaller numbers in k range as they are useless
            while (!q.isEmpty() && a[q.peekLast()] < a[i]) {
                q.pollLast();
            }
            // q contains index... r contains content
            q.offer(i);
            if (i >= k - 1) {
                r[ri++] = a[q.peek()];
            }
        }
        return r;
    }
```

```java
public List<List<Integer>> palindromePairs(String[] words) {
    List<List<Integer>> ret = new ArrayList<>();
    if (words == null || words.length < 2) return ret;
    Map<String, Integer> map = new HashMap<String, Integer>();
    for (int i=0; i<words.length; i++) map.put(words[i], i);
    for (int i=0; i<words.length; i++) {
        // System.out.println(words[i]);
        for (int j=0; j<=words[i].length(); j++) { // notice it should be "j <= words[i].length()"
            String str1 = words[i].substring(0, j);
            String str2 = words[i].substring(j);
            if (isPalindrome(str1)) {
                String str2rvs = new StringBuilder(str2).reverse().toString();
                if (map.containsKey(str2rvs) && map.get(str2rvs) != i) {
                    List<Integer> list = new ArrayList<Integer>();
                    list.add(map.get(str2rvs));
                    list.add(i);
                    ret.add(list);
                    // System.out.printf("isPal(str1): %s\n", list.toString());
                }
            }
            if (isPalindrome(str2)) {
                String str1rvs = new StringBuilder(str1).reverse().toString();
                // check "str.length() != 0" to avoid duplicates
                if (map.containsKey(str1rvs) && map.get(str1rvs) != i && str2.length()!=0) {
                    List<Integer> list = new ArrayList<Integer>();
                    list.add(i);
                    list.add(map.get(str1rvs));
                    ret.add(list);
                    // System.out.printf("isPal(str2): %s\n", list.toString());
                }
            }
        }
    }
    return ret;
}

private boolean isPalindrome(String str) {
    int left = 0;
    int right = str.length() - 1;
    while (left <= right) {
        if (str.charAt(left++) !=    str.charAt(right--)) return false;
```

```
        }
        return true;
    }
```